Problem Statement

Develop a Recommender System to enhance user experience by suggesting personalized movie recommendations. Utilizing collaborative filtering techniques such as item-based and user-based approaches, alongside Pearson correlation and nearest neighbors using cosine similarity, the system identifies similar users and their rated movies. Through matrix factorization, it distills latent features for more accurate predictions, offering users tailored movie suggestions aligned with their preferences and those of like-minded individuals.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in c:\users\gyanp\anaconda3\lib\sit
e-packages (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in c:\users\gyanp\anaconda3\lib\site-
packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in c:\users\gyanp\anaconda3\lib\site-
packages (from scikit-surprise) (1.26.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\gyanp\anaconda3\lib\site-p
ackages (from scikit-surprise) (1.12.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
from sklearn.impute import KNNImputer
from sklearn.preprocessing import StandardScaler
from surprise import KNNWithMeans
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise import Reader
from surprise import SVD
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from sklearn.manifold import TSNE
from scipy.sparse import csr_matrix
```

In [4]:
```
user = pd.read_fwf(r"C:\Users\gyanp\Downloads\zee-users.dat",encoding="ISO-8859-1")
user.head(2)
```

Out[4]:

	UserID::Gender::Age::Occupation::Zip-code
0	1::F::1::10::48067
1	2::M::56::16::70072

In [5]:
```
rating = pd.read_fwf(r"C:\Users\gyanp\Downloads\zee-ratings.dat",encoding="ISO-8859
rating.head(2)
```

Out[5]:

	UserID::MovieID::Rating::Timestamp
0	1::1193::5::978300760
1	1::661::3::978302109

In [6]:
```python
movie = pd.read_fwf(r"C:\Users\gyanp\Downloads\zee-movies.dat",encoding="ISO-8859-1
display(movie.head(2))
# Two irrelevant columns are there, Need to drop it
movie.drop(columns = ['Unnamed: 1','Unnamed: 2'], inplace = True)
movie.head(2)
```

	Movie ID::Title::Genres	Unnamed: 1	Unnamed: 2
0	1::Toy Story (1995)::Animation|Children's|Comedy	NaN	NaN
1	2::Jumanji (1995)::Adventure|Children's|Fantasy	NaN	NaN

Out[6]:

	Movie ID::Title::Genres
0	1::Toy Story (1995)::Animation|Children's|Comedy
1	2::Jumanji (1995)::Adventure|Children's|Fantasy

In [7]:
```python
# Display shape of all the datasets

user.shape, rating.shape, movie.shape
```

Out[7]:
```
((6040, 1), (1000209, 1), (3883, 1))
```

Data Cleaning

In [8]:
```python
def split_column(df, column, delimiter, column_names):
    # Split the column based on the delimiter
    split_data = df[column].str.split(delimiter, expand=True)
    # Assign new column names
    split_data.columns = column_names
    return split_data
```

Data cleaning of User dataframe

In [9]:
```python
# data cleaning of user data
# Define column names for the split columns
column_names = ['user_id', 'gender', 'age', 'occupation', 'zipcode']
# Apply the function to split the column
user_data = split_column(user, 'UserID::Gender::Age::Occupation::Zip-code', '::', c
user_data.head(2)
```

Out[9]:

	user_id	gender	age	occupation	zipcode
0	1	F	1	10	48067
1	2	M	56	16	70072

In [10]:
```python
# columns have been splitted. Need to analyse each dataset in detail.
print("distinct number of users:",user_data['user_id'].nunique())
print("distinct number of categories in age:", user_data['age'].nunique())
print("distinct number of categories in occupation:", user_data['occupation'].nuniq
print("-------------------------------")
display(user_data.info())
# There are 6040 unique users.
# Gender is denoted by a "M" for male and "F" for female
####### Age is chosen from the following ranges:
# 1: "Under 18",
```

```
# 18: "18-24",
# 25: "25-34",
# 35: "35-44",
# 45: "45-49",
# 50: "50-55",
# 56: "56+"
####### Occupation is chosen from the following choices:
# 0: "other" or not specified
# 1: "academic/educator"
# 2: "artist"
# 3: "clerical/admin"
# 4: "college/grad student"
# 5: "customer service"
# 6: "doctor/health care"
# 7: "executive/managerial"
# 8: "farmer"
# 9: "homemaker"
# 10: "K-12 student"
# 11: "lawyer"
# 13: "retired"
# 14: "sales/marketing"
# 15: "scientist"
# 16: "self-employed"
# 17: "technician/engineer"
# 18: "tradesman/craftsman"
# 19: "unemployed"
# 20: "writer"
########  zipcode should be int. If this feature turns out to be relevant, convert
```

```
distinct number of users: 6040
distinct number of categories in age: 7
distinct number of categories in occupation: 21
-------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   user_id     6040 non-null   object
 1   gender      6040 non-null   object
 2   age         6040 non-null   object
 3   occupation  6040 non-null   object
 4   zipcode     6040 non-null   object
dtypes: object(5)
memory usage: 236.1+ KB
None
```

Data Cleaning of rating datafrane

In [11]:

```
# data cleaning of rating data
# Define column names for the split columns
column_names = ['user_id', 'movie_id', 'rating', 'timestamp']
# Apply the function to split the column
rating_data = split_column(rating, 'UserID::MovieID::Rating::Timestamp', '::', colu
rating_data.head(2)
```

Out[11]:

	user_id	movie_id	rating	timestamp
0	1	1193	5	978300760
1	1	661	3	978302109

In [12]:
```
display(rating_data.info())
# UserIDs range between 1 and 6040
# MovieIDs range between 1 and 3952
# Ratings are made on a 5-star scale (whole-star ratings only)
# Timestamp is represented in seconds
# Each user has at least 20 ratings (given)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   user_id    1000209 non-null  object
 1   movie_id   1000209 non-null  object
 2   rating     1000209 non-null  object
 3   timestamp  1000209 non-null  object
dtypes: object(4)
memory usage: 30.5+ MB
None
```

In [13]:
```
# Timestamp needs to convert in hours
import datetime
rating_data['timestamp'] = pd.to_datetime(rating_data['timestamp'], unit='s')
#rating_data['timestamp'] = rating_data['timestamp'].astype('int')
#rating_data['hour'] = rating_data['timestamp'].apply(lambda x: datetime.datetime.f
display(rating_data.head())
```

	user_id	movie_id	rating	timestamp
0	1	1193	5	2000-12-31 22:12:40
1	1	661	3	2000-12-31 22:35:09
2	1	914	3	2000-12-31 22:32:48
3	1	3408	4	2000-12-31 22:04:35
4	1	2355	5	2001-01-06 23:38:11

In [14]:
```
rating_data['user_id'].value_counts()
# user_id 4169 has rated maximum movies, that is, 2314
# minimum 20 movies have been rated by each user.
```

Out[14]:
```
4169    2314
1680    1850
4277    1743
1941    1595
1181    1521
        ... 
5725      20
3407      20
1664      20
4419      20
3021      20
Name: user_id, Length: 6040, dtype: int64
```

In [15]:
```
rating_data['movie_id'].value_counts()
# movie_id 2858 got rated maximum times, that is, 3428
# each movie has rated atleast once.
```

Out[15]:
```
2858    3428
260     2991
1196    2990
1210    2883
480     2672
        ... 
3458       1
2226       1
1815       1
398        1
2909       1
Name: movie_id, Length: 3706, dtype: int64
```

Data Cleaning of movie datafrane

In [16]:
```
# data cleaning of movie data
# Define column names for the split columns
column_names = ['movie_id', 'title', 'genres']
# Apply the function to split the column
movie_data = split_column(movie, 'Movie ID::Title::Genres', '::', column_names)
movie_data.head(2)
```

Out[16]:

	movie_id	title	genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy

In [17]:
```
display(movie_data.info())
# Titles are identical to titles provided by the IMDB (including year of release)
# Genres are pipe-separated and are selected from the following genres: Action, Adv
# Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery,
# and Western
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   movie_id  3883 non-null   object
 1   title     3883 non-null   object
 2   genres    3858 non-null   object
dtypes: object(3)
memory usage: 91.1+ KB
None
```

deriving new features like ‘Release Year’ and "movie_name"

In [18]:
```
movie_data['release_year'] = movie_data['title'].str[-5:-1]
movie_data['movie_name'] = movie_data['title'].str[:-7]
movie_data
```

Out[18]:

	movie_id	title	genres	release_year	movie_name
0	1	Toy Story (1995)	Animation|Children's|Comedy	1995	Toy Story
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	1995	Jumanji
2	3	Grumpier Old Men (1995)	Comedy|Romance	1995	Grumpier Old Men
3	4	Waiting to Exhale (1995)	Comedy|Drama	1995	Waiting to Exhale
4	5	Father of the Bride Part II (1995)	Comedy	1995	Father of the Bride Part II
...	...	...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy	2000	Meet the Parents
3879	3949	Requiem for a Dream (2000)	Drama	2000	Requiem for a Dream
3880	3950	Tigerland (2000)	Drama	2000	Tigerland
3881	3951	Two Family House (2000)	Drama	2000	Two Family House
3882	3952	Contender, The (2000)	Drama|Thriller	2000	Contender, The

3883 rows × 5 columns

exploding genres values and create clean dataset

In [19]:
```
movie_data['genres'] = movie_data['genres'].str.split('|')
movie_data = movie_data.explode('genres')
```

In [20]:
```
#movie_data = movie_data[['movie_id','movie_name','genres','release_year']]
movie_data
```

Out[20]:

	movie_id	title	genres	release_year	movie_name
0	1	Toy Story (1995)	Animation	1995	Toy Story
0	1	Toy Story (1995)	Children's	1995	Toy Story
0	1	Toy Story (1995)	Comedy	1995	Toy Story
1	2	Jumanji (1995)	Adventure	1995	Jumanji
1	2	Jumanji (1995)	Children's	1995	Jumanji
...	...	...	...	...	...
3879	3949	Requiem for a Dream (2000)	Drama	2000	Requiem for a Dream
3880	3950	Tigerland (2000)	Drama	2000	Tigerland
3881	3951	Two Family House (2000)	Drama	2000	Two Family House
3882	3952	Contender, The (2000)	Drama	2000	Contender, The
3882	3952	Contender, The (2000)	Thriller	2000	Contender, The

6366 rows × 5 columns

In [21]:
```python
movie_data['genres'].nunique()
```

Out[21]:
```
63
```

In [22]:
```python
movie_data['genres'].unique()

# It can be observed that genres haven't been labelled correctly to movies.
# Children category can be seen as  "Children's", 'Chil', 'Childre', 'Childr','Chil
# Similarly, Romance category labelled differently as 'Rom','Ro', 'Roman', 'R', 'Ro
# Similarly, Comedy category labelled differently as 'Come', 'Comed', 'Com'
# Similarly Animation is also written as 'Animati'
# Similarly, Adventure also written as 'Adv','Adventu','Adventur','Advent'
# Similarly, Fantasy also written as  'Fantas', 'Fant', 'F'
# Similarly, Action is also written as 'Acti'
# Similarly, Thriller written as 'Th', 'Thri','Thrille'
# 'D', 'S', 'A' and '' are unknown genres. So, label them as None
```

Out[22]:
```
array(['Animation', "Children's", 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', None,
       'Film-Noir', 'Dram', 'Western', 'Chil', '', 'Fantas', 'Dr', 'D',
       'Documenta', 'Wester', 'Fant', 'Music', 'Childre', 'Childr', 'Rom',
       'Animati', 'Children', 'Come', "Children'", 'Sci-F', 'Adv',
       'Adventu', 'Horro', 'Docu', 'S', 'Sci-', 'Document', 'Th', 'Roman',
       'Documen', 'We', 'F', 'Ro', 'R', 'Sci', 'Chi', 'Thri', 'Adventur',
       'Advent', 'Acti', 'Roma', 'A', 'Comed', 'Com', 'Thrille', 'Wa',
       'Horr'], dtype=object)
```

In [23]:
```python
genres_mapping = { "Children's": 'Children','Chil': 'Children', 'Childre': 'Childre
     'Chi': 'Children','Children': 'Children',"Children'": 'Children','Rom': 'Romanc
     'Roman': 'Romance','R': 'Romance', 'Roma': 'Romance', 'Come': 'Comedy','Comed':
     'Animati': 'Animation','Adv': 'Adventure','Adventu': 'Adventure','Adventur': 'A
     'Advent': 'Adventure','Fantas': 'Fantasy','Fant': 'Fantasy','F': 'Fantasy', 'Ac
     'Th': 'Thriller','Thri': 'Thriller','Thrille': 'Thriller', 'Sci-Fi,':'Sci-Fi','
     'Sci-':'Sci-Fi', 'Sci': 'Sci-Fi,','S': 'Sci-Fi,', 'Docu': 'Documentary', 'Docum
     'Document': 'Documentary','Documen':'Documentary' ,'Wa': 'War', 'Horro':'Horror
     'Wester':'Western', 'Dram':'Drama','Music':'Musical','Dr':'Drama', 'We':'Wester
                    'D':None, 'A': None}

# Apply the mapping to correct errors in the 'genres' column
movie_data['genres'] = movie_data['genres'].apply(lambda x: genres_mapping.get(x, x
```

In [24]:
```python
movie_data['genres'].unique()
```

Out[24]:
```
array(['Animation', 'Children', 'Comedy', 'Adventure', 'Fantasy',
       'Romance', 'Drama', 'Action', 'Crime', 'Thriller', 'Horror',
       'Sci-Fi', 'Documentary', 'War', 'Musical', 'Mystery', None,
       'Film-Noir', 'Western', 'Sci-Fi,'], dtype=object)
```

In [25]:
```python
movie_data.shape
# earlier rows were 3883 and after exploding, rows now 6366
```

Out[25]:
```
(6366, 5)
```

Missing value check and treatment

In [26]:
```python
print("Missing values in user_data is:",user_data.isna().sum().sum())
print("-------------------------------")
print("Missing values in rating_data is:",rating_data.isna().sum().sum())
print("-------------------------------")
print("Missing values in movie_data is:",movie_data.isna().sum().sum())
```

```
print("% data missing in movie_dataset is: ",((movie_data.isna().sum().sum())/movie
# there are missing values in movie_dataset
print("------------------------------")
display(movie_data.isna().sum())
```

```
Missing values in user_data is: 0
------------------------------
Missing values in rating_data is: 0
------------------------------
Missing values in movie_data is: 38
% data missing in movie_dataset is:  0.5969211435752434
------------------------------
movie_id         0
title            0
genres          38
release_year     0
movie_name       0
dtype: int64
```

In [27]:
```
# There are 25 missing values in genres feature of this movie dataset
# As 0.6% data is missing, which is too small, rows can be dropped.
movie_data = movie_data.loc[~movie_data['genres'].isna()]
print("Now, the missing value in movie_dataset is: ",movie_data.isna().sum().sum())
```

```
Now, the missing value in movie_dataset is:  0
```

In [28]:
```
#movie_data = movie_data[['movie_id','movie_name','genres','release_year']]
```

Merging the data files into one single dataframe

In [29]:
```
df = pd.merge(pd.merge(movie_data,rating_data,left_on = 'movie_id',right_on='movie_
              user_data,on='user_id',how='inner')
df
```

Out[29]:

	movie_id	title	genres	release_year	movie_name	user_id	rating	timestamp
0	1	Toy Story (1995)	Animation	1995	Toy Story	1	5	2001-01-06 23:37:48
1	1	Toy Story (1995)	Children	1995	Toy Story	1	5	2001-01-06 23:37:48
2	1	Toy Story (1995)	Comedy	1995	Toy Story	1	5	2001-01-06 23:37:48
3	48	Pocahontas (1995)	Animation	1995	Pocahontas	1	5	2001-01-06 23:39:11
4	48	Pocahontas (1995)	Children	1995	Pocahontas	1	5	2001-01-06 23:39:11
...	...	...	...	...	...	...	...	...
2057303	3536	Keeping the Faith (2000)	Romance	2000	Keeping the Faith	5727	5	2000-05-16 15:11:42
2057304	3555	U-571 (2000)	Action	2000	U-571	5727	3	2000-05-16 15:24:59
2057305	3555	U-571 (2000)	Thriller	2000	U-571	5727	3	2000-05-16 15:24:59
2057306	3578	Gladiator (2000)	Action	2000	Gladiator	5727	5	2000-05-16 15:16:11
2057307	3578	Gladiator (2000)	Drama	2000	Gladiator	5727	5	2000-05-16 15:16:11

2057308 rows × 12 columns

In [30]:

```
# checking the structure & characteristics of the dataset \
print(df.shape)
print("--------------------------------------------")
print(df.info())

# New dataset has 2060031 rows with 11 features
```

```
(2057308, 12)
-------------------------------------------
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2057308 entries, 0 to 2057307
Data columns (total 12 columns):
 #   Column        Dtype         
---  ------        -----         
 0   movie_id      object        
 1   title         object        
 2   genres        object        
 3   release_year  object        
 4   movie_name    object        
 5   user_id       object        
 6   rating        object        
 7   timestamp     datetime64[ns]
 8   gender        object        
 9   age           object        
 10  occupation    object        
 11  zipcode       object        
dtypes: datetime64[ns](1), object(11)
memory usage: 204.0+ MB
None
```

In [31]:
```
df.isna().sum().sum()
# There are no missing values in single dataframe
```

Out[31]:
```
0
```

In [32]:
```
df.head(2)
```

Out[32]:

	movie_id	title	genres	release_year	movie_name	user_id	rating	timestamp	gender	ag
0	1	Toy Story (1995)	Animation	1995	Toy Story	1	5	2001-01-06 23:37:48	F	
1	1	Toy Story (1995)	Children	1995	Toy Story	1	5	2001-01-06 23:37:48	F	

Necessary type conversions

In [33]:
```
df['release_year'] = df['release_year'].astype('int')
df['rating'] = df['rating'].astype('int')
```

Statistical analysis of data

In [34]:
```
df.describe()

# mean of rating of all data is 3.57.
# minimum rating is 1 and maximum rating is 1.
# 25% of ratings are under 3 in the dataset
# 50% and 75% of ratings are under rating 4 of whole dataset
```

Out[34]:

	release_year	rating
count	2.057308e+06	2.057308e+06
mean	1.986811e+03	3.575766e+00
std	1.412354e+01	1.116257e+00
min	1.919000e+03	1.000000e+00
25%	1.983000e+03	3.000000e+00
50%	1.992000e+03	4.000000e+00
75%	1.997000e+03	4.000000e+00
max	2.000000e+03	5.000000e+00

In [35]:

```
display(df.describe(include='object'))
# there are 3677 unique movie_ids and 3640 unique movie names. This infer that few
# there are distinct 19 genres of movies in dataset
# top movie is Men in Black, top genre is comedy, top rating is 4.
```

	movie_id	title	genres	movie_name	user_id	gender	age	occupation	zipcod
count	2057308	2057308	2057308	2057308	2057308	2057308	2057308	2057308	205730
unique	3677	3677	19	3635	6040	2	7	21	34
top	1580	Men in Black (1997)	Comedy	Men in Black	4169	M	25	4	941
freq	10152	10152	353555	10152	3966	1561317	814006	271499	76

In [36]:

```
df['release_year'].min(), df['release_year'].max()

# The movie release years spans from 1919 to 2000
```

Out[36]:

```
(1919, 2000)
```

Exploratory Data Analysis

In [37]:

```
# Drop duplicate user_id rows to ensure each user is counted only once
unique_users = df.drop_duplicates(subset='user_id')

# Calculate unique count of males and females
gender_counts = unique_users['gender'].value_counts()
age_counts = unique_users['age'].value_counts()
occupation_counts = unique_users['occupation'].value_counts()
rating_counts = unique_users['rating'].value_counts()
```

In [38]:

```
gender_counts
```

Out[38]:

```
M    4331
F    1709
Name: gender, dtype: int64
```

In [39]:

```
age_counts
```

Out[39]:
```
25    2096
35    1193
18    1103
45     550
50     496
56     380
1      222
Name: age, dtype: int64
```

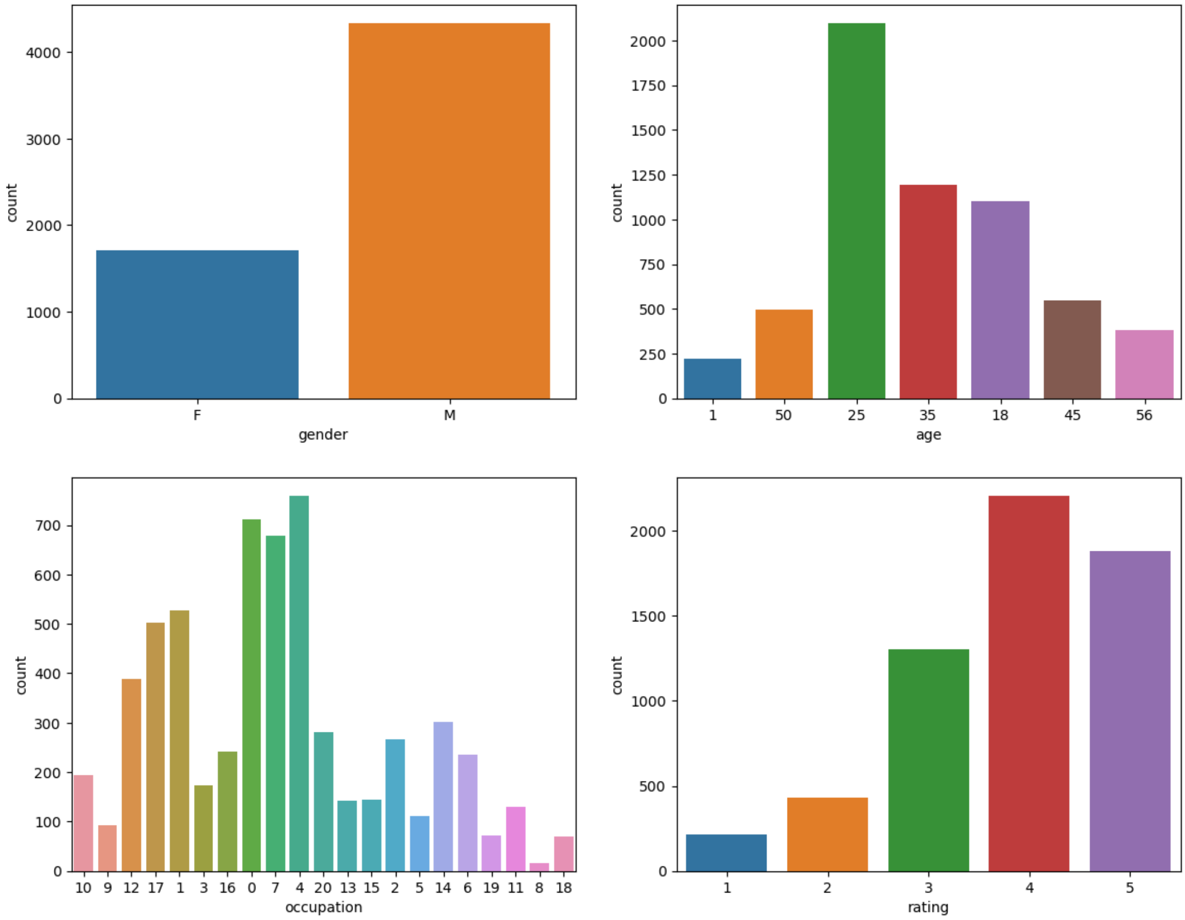
In [40]:
```
occupation_counts
```

Out[40]:
```
4     759
0     711
7     679
1     528
17    502
12    388
14    302
20    281
2     267
16    241
6     236
10    195
3     173
15    144
13    142
11    129
5     112
9      92
19     72
18     70
8      17
Name: occupation, dtype: int64
```

In [41]:
```
rating_counts
```

Out[41]:
```
4    2205
5    1883
3    1306
2     432
1     214
Name: rating, dtype: int64
```

In [42]:
```
plt.figure(figsize=(14,11))
plt.subplot(2,2,1)
sns.countplot(x='gender', data=unique_users)
plt.subplot(2,2,2)
sns.countplot(x='age', data=unique_users)
plt.subplot(2,2,3)
sns.countplot(x='occupation', data=unique_users)
plt.subplot(2,2,4)
sns.countplot(x='rating', data=unique_users)
plt.show()
```



1. total males are 4331 and total females are 1709
2. Majority users (2096 users) belongs to age criteria 25 to 34, followed by 1193 users belong to age group 35-44, and 1103 users belong to age group 18-24
3. Maximum users (759 users) are college/grad student and minimum users (only 17 users) are farmers.
4. Mostly users(2205 users) have rated movies as 4 followed by 1883 users rated as 5.
5. 214 users have rated movies as 1.

Questionnaire 1: Users of which age group have watched and rated the most number of movies?

- *Users of age group 25-34 have watched and rated most number of movies*

Questionnaire 2: Users belonging to which profession have watched and rated the most movies?

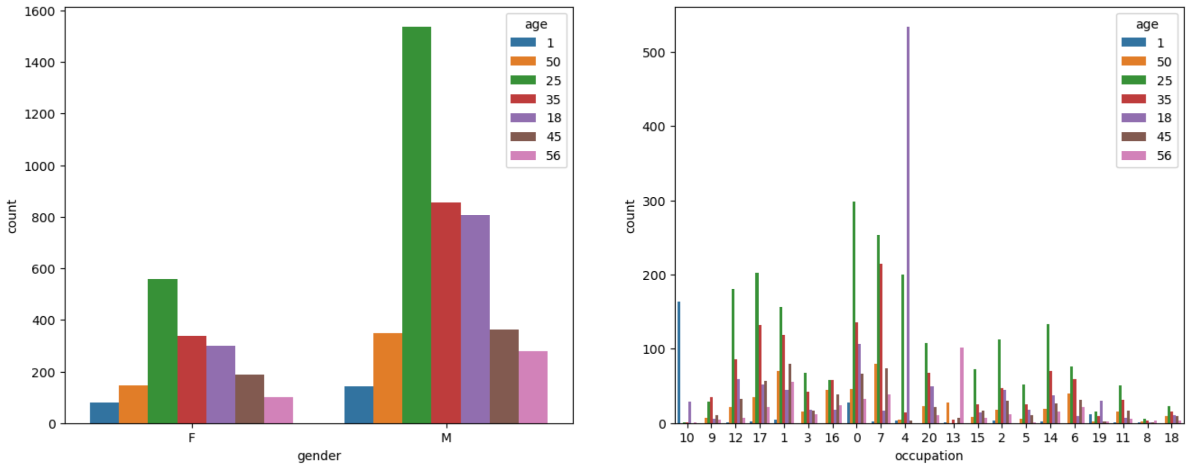
- *Users belonging to college/graduate student have watched and rated the most movies.*

Questionnaire 3: Most of the users in our dataset who've rated the movies are Male.

- **True**, *total males are 4331 and total females are 1709*

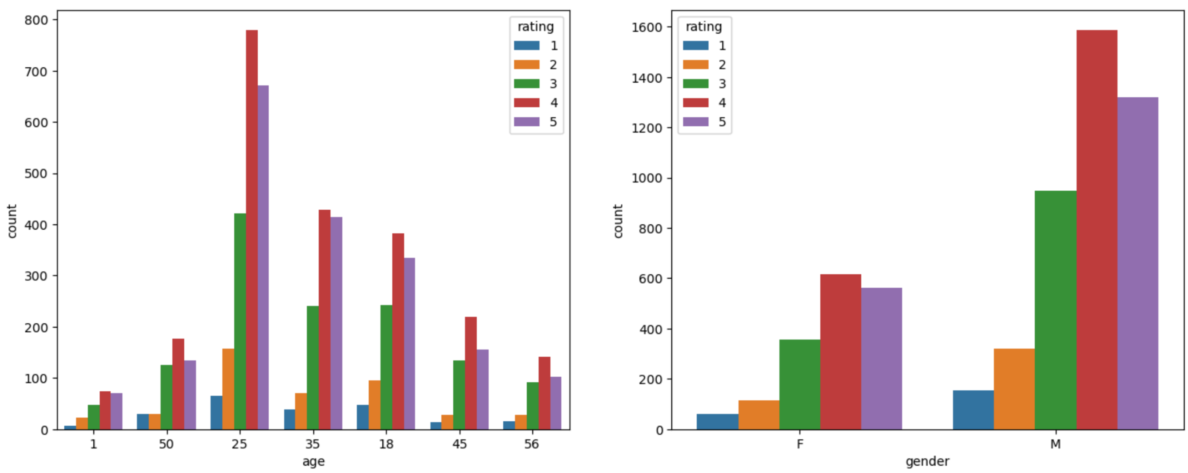
In [43]:
```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.countplot(data=unique_users, x='gender', hue='age')
plt.subplot(1,2,2)
```

```
sns.countplot(data=unique_users, x='occupation', hue='age')
plt.show()
```



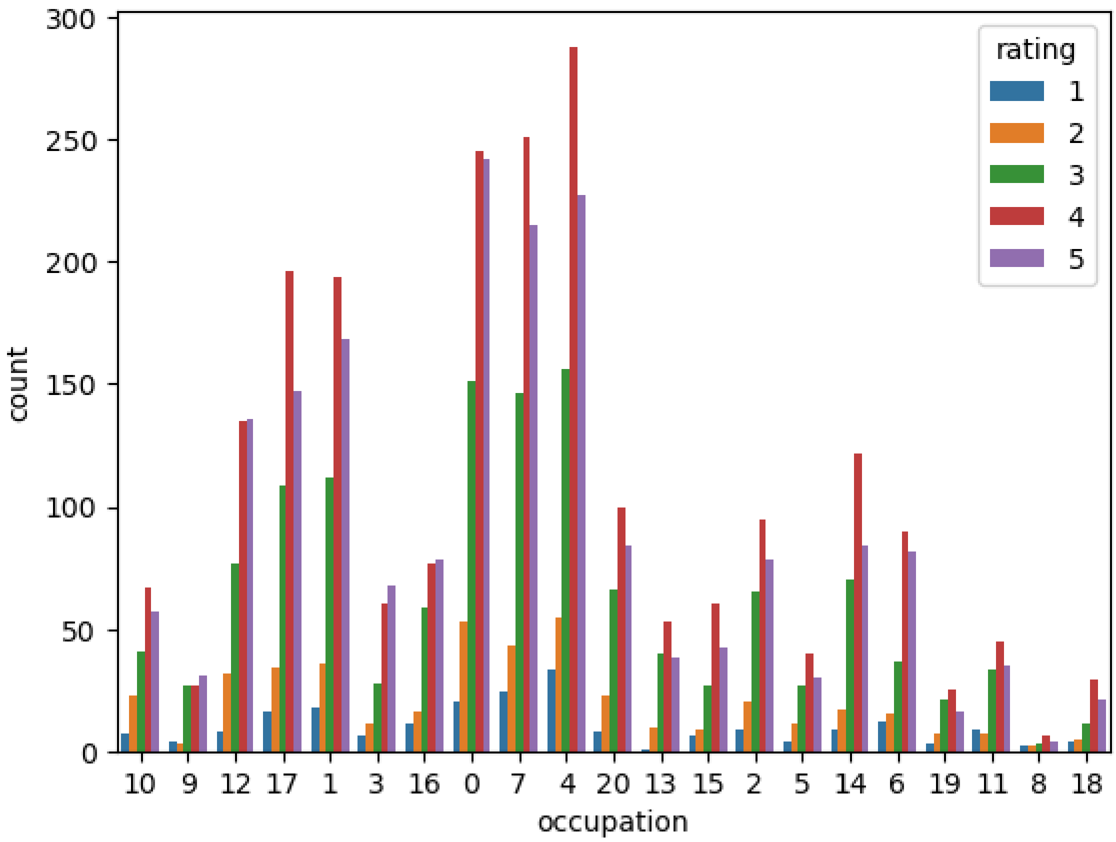
1. Majority males and females who belong to age group 25 to 34 years have watched and rated movies.
2. very few males and females below 18 years of age have watched and rated movies
3. Age group with 18-24 college/grad students watched movies a lot and rated them as well.

In [44]:
```
plt.figure(figsize=(16,6))
plt.subplot(1,2,1)
sns.countplot(data=unique_users, x='age', hue='rating')
plt.subplot(1,2,2)
sns.countplot(data=unique_users, x='gender', hue='rating')
plt.show()
```



1. As we can observe that age group of 25-34 rated movies very actively and almost each age group rated 4 frequently.
2. In age segregation also, one can observe that irrespective of gender, people rated 4 and 5 frequently.

In [45]:
```
sns.countplot(data=unique_users, x='occupation', hue='rating')
plt.show()
```



1. users with occupation 0- other,4- college/grad student,7- executive/managerial and 17- technician/engineer are likely to involve in movies and ratings.
2. users with occupation 8- farmer are less likely to engage in watching and rating movies.

In [46]:
```
# Drop duplicate user_id rows to ensure each user is counted only once
unique_movies = df.drop_duplicates(subset='movie_id')
unique_movies
# Calculate unique count of males and females
genre_counts = unique_movies['genres'].value_counts()
rating_counts_ = unique_movies['rating'].value_counts()
release_year_counts = unique_movies['release_year'].value_counts()
```

In [47]:
```
genre_counts
```

Out[47]:
```
Drama          1069
Comedy          981
Action          493
Horror          257
Adventure       153
Crime           123
Documentary     103
Thriller        100
Animation        90
Children         89
Romance          45
Sci-Fi           44
Mystery          35
Western          32
Musical          25
Film-Noir        25
War              11
Fantasy           2
Name: genres, dtype: int64
```

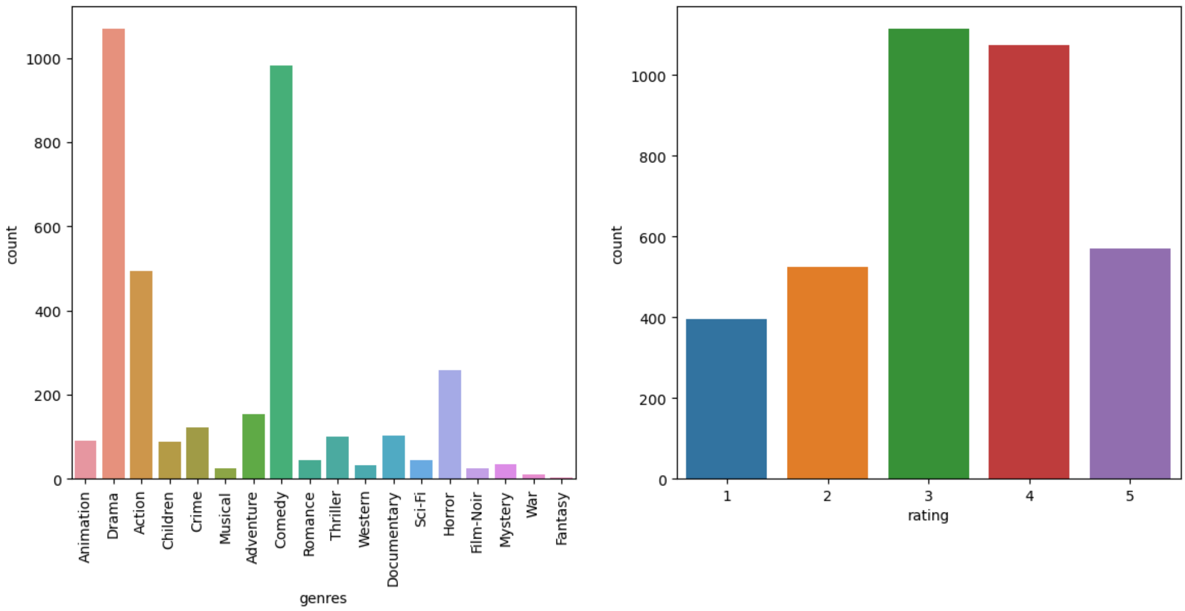
In [48]:
```
rating_counts_
```

Out[48]:
```
3    1114
4    1073
5     569
2     525
1     396
Name: rating, dtype: int64
```

In [49]:
```
release_year_counts
```

Out[49]:
```
1998    311
1996    311
1995    309
1997    304
1999    271
       ...
1928      2
1929      2
1922      1
1921      1
1920      1
Name: release_year, Length: 81, dtype: int64
```

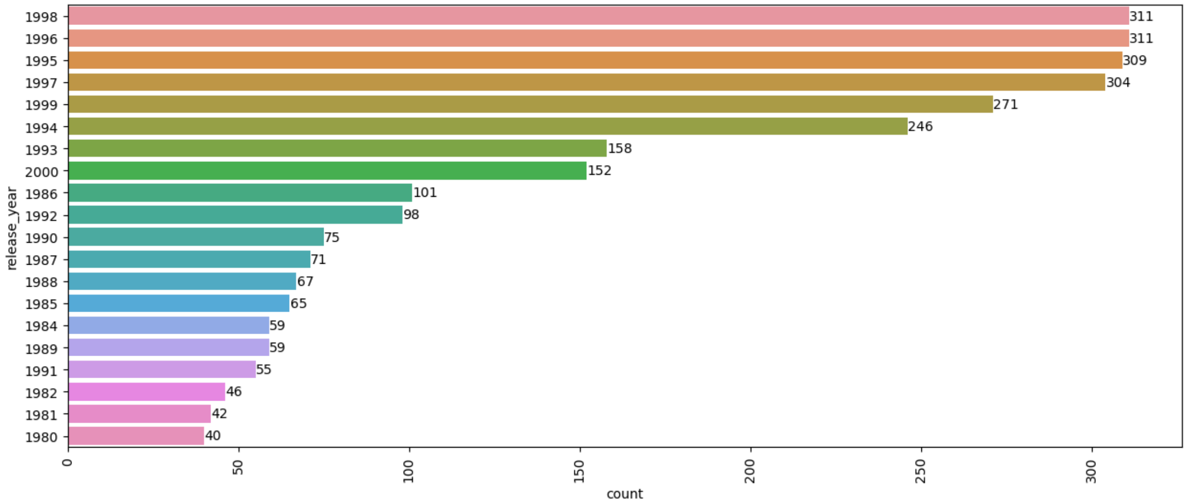
In [50]:
```
plt.figure(figsize=(14,6))
plt.subplot(1,2,1)
sns.countplot(x='genres', data=unique_movies)
plt.xticks(rotation = 90)
plt.subplot(1,2,2)
sns.countplot(x='rating', data=unique_movies)
plt.show()
```



1. The most popular genre is Drama having 1069 movies and fantasy genre has just two movies
2. 1114 movies were rated as 3, followed by 1073 movies rated as 4.
3. 396 movies were rated as 1.

In [51]:
```
plt.figure(figsize=(15,6))
sns.countplot(y='release_year', data=unique_movies, order= unique_movies['release_y
for i, count in enumerate(unique_movies['release_year'].value_counts().iloc[:20]):
    plt.text(count + 0.1, i, str(count), va='center')
```

```
plt.xticks(rotation = 90)
plt.show()
```



311 movies released in year 1996 and 1998 followed by 309 in 1995 and 304 in 1997

In [52]:
```
# Create bins for each decade
decade_bins = [1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980, 1990, 2000]

# Create labels for each decade
decade_labels = ['10s', '20s', '30s', '40s', '50s', '60s', '70s', '80s', '90s']

# Bin the 'release_year' data into decade bins
unique_movies['decade'] = pd.cut(unique_movies['release_year'], bins=decade_bins, l

# Count the number of movies released in each decade
decade_counts = unique_movies['decade'].value_counts().sort_index()
decade_counts
```

```
C:\Users\gyanp\AppData\Local\Temp\ipykernel_20504\2640747710.py:8: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy
  unique_movies['decade'] = pd.cut(unique_movies['release_year'], bins=decade_bin
s, labels=decade_labels, right=False)
```

Out[52]:
```
10s       3
20s      23
30s      71
40s     120
50s     164
60s     184
70s     237
80s     585
90s    2138
Name: decade, dtype: int64
```

Questionnaire 4. Most of the movies present in our dataset were released in which decade?

1. 70s b. 90s c. 50s d.80s

- **b. 90s** *,2138 movies released during this decade, followed by 585 movies in 80s*

In [53]:
```
movie_user_rating = df[["movie_id","movie_name","user_id","rating"]]
movie_user_ratings = movie_user_rating.drop_duplicates()
movie_user_counts = movie_user_ratings.groupby('movie_name')['user_id'].nunique().s
movie_user_counts[:5]
```

Out[53]:
```
movie_name
American Beauty                                     3428
Star Wars: Episode IV - A New Hope                  2991
Star Wars: Episode V - The Empire Strikes Back      2990
Star Wars: Episode VI - Return of the Jedi          2883
Jurassic Park                                       2672
Name: user_id, dtype: int64
```

Questionnaire 5 : The movie with maximum no. of ratings is ___.

- **American Beauty**

Group the data according to the average rating and no. of ratings

In [54]:
```
df_1 = df.copy()
```

In [55]:
```
df_1
```

Out[55]:

	movie_id	title	genres	release_year	movie_name	user_id	rating	timestamp
0	1	Toy Story (1995)	Animation	1995	Toy Story	1	5	2001-01-06 23:37:48
1	1	Toy Story (1995)	Children	1995	Toy Story	1	5	2001-01-06 23:37:48
2	1	Toy Story (1995)	Comedy	1995	Toy Story	1	5	2001-01-06 23:37:48
3	48	Pocahontas (1995)	Animation	1995	Pocahontas	1	5	2001-01-06 23:39:11
4	48	Pocahontas (1995)	Children	1995	Pocahontas	1	5	2001-01-06 23:39:11
...	...	...	...	...	...	...	...	...
2057303	3536	Keeping the Faith (2000)	Romance	2000	Keeping the Faith	5727	5	2000-05-16 15:11:42
2057304	3555	U-571 (2000)	Action	2000	U-571	5727	3	2000-05-16 15:24:59
2057305	3555	U-571 (2000)	Thriller	2000	U-571	5727	3	2000-05-16 15:24:59
2057306	3578	Gladiator (2000)	Action	2000	Gladiator	5727	5	2000-05-16 15:16:11
2057307	3578	Gladiator (2000)	Drama	2000	Gladiator	5727	5	2000-05-16 15:16:11

2057308 rows × 12 columns

In [56]:
```
# Remove duplicates caused by genre explosion
unique_movies = df_1.drop_duplicates(subset=['movie_id', 'user_id'])
display(unique_movies.head())
print("----------------------------------------------------------------------------------

# Group by 'movie_id' and 'movie_name' and calculate average rating and number of r
movie_grouped = unique_movies.groupby(['movie_id', 'movie_name']).agg({'rating': ['

# Rename the columns for clarity
movie_grouped.columns = ['average_rating', 'num_ratings']

# Reset index to make the grouped columns accessible
movie_grouped.reset_index(inplace=True)
movie_grouped_data = pd.DataFrame(movie_grouped)

movie_grouped_data.head()
```

	movie_id	title	genres	release_year	movie_name	user_id	rating	timestamp	gend
0	1	Toy Story (1995)	Animation	1995	Toy Story	1	5	2001-01-06 23:37:48	
3	48	Pocahontas (1995)	Animation	1995	Pocahontas	1	5	2001-01-06 23:39:11	
7	150	Apollo 13 (1995)	Drama	1995	Apollo 13	1	5	2000-12-31 22:29:37	
8	260	Star Wars: Episode IV - A New Hope (1977)	Action	1977	Star Wars: Episode IV - A New Hope	1	4	2000-12-31 22:12:40	
11	527	Schindler's List (1993)	Drama	1993	Schindler's List	1	5	2001-01-06 23:36:35	

```
--------------------------------------------------------------------------------
----------------------------
```

Out[56]:

	movie_id	movie_name	average_rating	num_ratings
0	1	Toy Story	4.146846	2077
1	10	GoldenEye	3.540541	888
2	100	City Hall	3.062500	128
3	1000	Curdled	3.050000	20
4	1002	Ed's Next Move	4.250000	8

Analysis done on grouping of data based on average rating

In [57]:
```
sorted_by_average_rating = movie_grouped_data.sort_values(by='average_rating', asce
sorted_by_average_rating.head()
```

Out[57]:

	movie_id	movie_name	average_rating	num_ratings
2350	3280	Baby, The	5.0	1
2747	3656	Lured	5.0	1
2698	3607	One Little Indian	5.0	1
2459	3382	Song of Freedom	5.0	1
803	1830	Follow the Bitch	5.0	1

In [58]:

```python
# Define rating categories
def get_rating_category(avg_rating):
    if avg_rating >= 4.0:
        return "Best rated"
    elif 3.0 <= avg_rating < 4.0:
        return "Better rated"
    elif 2.0 <= avg_rating < 3.0:
        return "Average rated"
    elif 1.0 <= avg_rating < 2.0:
        return "Worst rated"
    else:
        return "Unknown"

# Apply the function to create a new column 'rating_category'
movie_grouped_data['avg_rating_category'] = movie_grouped_data['average_rating'].ap
movie_grouped_data
```

Out[58]:

	movie_id	movie_name	average_rating	num_ratings	avg_rating_category
0	1	Toy Story	4.146846	2077	Best rated
1	10	GoldenEye	3.540541	888	Better rated
2	100	City Hall	3.062500	128	Better rated
3	1000	Curdled	3.050000	20	Better rated
4	1002	Ed's Next Move	4.250000	8	Best rated
...	...	...	...	...	...
3672	994	Big Night	4.095556	450	Best rated
3673	996	Last Man Standing	2.906250	256	Average rated
3674	997	Caught	3.357143	28	Better rated
3675	998	Set It Off	3.010753	93	Better rated
3676	999	2 Days in the Valley	3.283217	286	Better rated

3677 rows × 5 columns

In [59]:

```python
# Group movies by rating category
grouped_by_rating_category = movie_grouped_data.groupby('avg_rating_category')

# Print the counts of movies in each rating category
for category, group in grouped_by_rating_category:
    print(f"{category}: {group.shape[0]} movies")
```

```
Average rated: 995 movies
Best rated: 426 movies
Better rated: 2099 movies
Worst rated: 157 movies
```

1. There are 426 movies who are best rated, that is, their average rating is above 4.
2. There are 2099 movies who are better, that is, their average rating is between 3 and 4.
3. There are 995 average rated movies, their average rating is between 2 and 3
4. 157 movies are labelled as worst rated movies. Their average rating is below 2.

Analysis done on grouping of data based on number of ratings

In [60]:
```
sorted_by_num_ratings = movie_grouped_data.sort_values(by='num_ratings', ascending=
sorted_by_num_ratings
```

Out[60]:

	movie_id	movie_name	average_rating	num_ratings	avg_rating_category
1903	2858	American Beauty	4.317386	3428	Best rated
1631	260	Star Wars: Episode IV - A New Hope	4.453694	2991	Best rated
189	1196	Star Wars: Episode V - The Empire Strikes Back	4.292977	2990	Best rated
206	1210	Star Wars: Episode VI - Return of the Jedi	4.022893	2883	Best rated
3158	480	Jurassic Park	3.763847	2672	Better rated
...	...	...	...	...	...
1030	2039	Cheetah	1.000000	1	Worst rated
433	1430	Underworld	1.000000	1	Worst rated
3345	658	Billy's Holiday	3.000000	1	Better rated
570	1579	For Ever Mozart	3.000000	1	Better rated
887	1908	Resurrection Man	3.000000	1	Better rated

3677 rows × 5 columns

In [61]:
```
print(sorted_by_num_ratings["num_ratings"].max())
print(sorted_by_num_ratings["num_ratings"].min())
```

```
3428
1
```

In [62]:
```
# Define rating categories
def get_rating_category(num_rating):
    if num_rating >= 2000:
        return "Grade A"
    elif 1000 <= num_rating < 2000:
        return "Grade B"
    elif 500 <= num_rating < 1000:
        return "Grade C"
    elif num_rating < 500:
        return "Grade D"
    else:
        return "Unknown"
```

```
# Apply the function to create a new column 'rating_category'
movie_grouped_data['count_rating_category'] = movie_grouped_data['num_ratings'].app
movie_grouped_data
```

Out[62]:

	movie_id	movie_name	average_rating	num_ratings	avg_rating_category	count_rating_categ
0	1	Toy Story	4.146846	2077	Best rated	Gra
1	10	GoldenEye	3.540541	888	Better rated	Gra
2	100	City Hall	3.062500	128	Better rated	Gra
3	1000	Curdled	3.050000	20	Better rated	Gra
4	1002	Ed's Next Move	4.250000	8	Best rated	Gra
...	...	...	...	...	...	
3672	994	Big Night	4.095556	450	Best rated	Gra
3673	996	Last Man Standing	2.906250	256	Average rated	Gra
3674	997	Caught	3.357143	28	Better rated	Gra
3675	998	Set It Off	3.010753	93	Better rated	Gra
3676	999	2 Days in the Valley	3.283217	286	Better rated	Gra

3677 rows × 6 columns

In [63]:

```
# Group movies by rating category
grouped_by_count_rating_category = movie_grouped_data.groupby('count_rating_categor

# Print the counts of movies in each rating category
for category, group in grouped_by_count_rating_category:
    print(f"{category}: {group.shape[0]} movies")
```

```
Grade A: 31 movies
Grade B: 175 movies
Grade C: 409 movies
Grade D: 3062 movies
```

1. There are 31 movies who are listed as grade A, that is, number of ratings received on these movies is above 2000.
2. There are 2175 movies who are categorize as grade B movies, that is, number of ratings received is between 1000 and 2000.
3. 409 movies are grade C movies, their count of rating is between 500 and 1000
4. 3062 movies are labelled as grade D movies. The number of ratings received on these movies is below 500.

creating features like average rating per user, average rating per movie, total number of ratings per movie

In [64]:

```
# Average rating per user
user_avg_rating = unique_movies.groupby('user_id')['rating'].mean()

# Average rating per movie
```

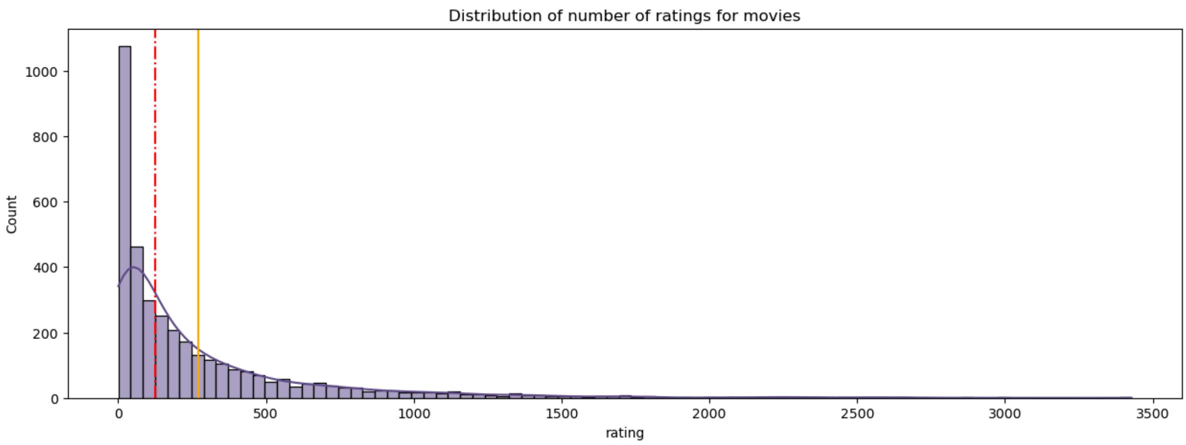
```python
movie_avg_rating = unique_movies.groupby('movie_id')['rating'].mean()

# Total number of ratings per movie
total_ratings_per_movie = unique_movies.groupby('movie_id')['rating'].count()

# Merge the features into a single DataFrame
features_df = pd.DataFrame({'user_avg_rating': user_avg_rating,'movie_avg_rating':
    'total_ratings_per_movie': total_ratings_per_movie
}).reset_index()

features_df
```

Out[64]:

	index	user_avg_rating	movie_avg_rating	total_ratings_per_movie
0	1	4.188679	4.146846	2077.0
1	10	4.120603	3.540541	888.0
2	100	3.026316	3.062500	128.0
3	1000	4.130952	3.050000	20.0
4	1001	3.651596	NaN	NaN
...	...	...	...	...
6035	995	3.897959	NaN	NaN
6036	996	3.935811	2.906250	256.0
6037	997	3.933333	3.357143	28.0
6038	998	4.118519	3.010753	93.0
6039	999	3.189781	3.283217	286.0

6040 rows × 4 columns

In [65]:
```python
import seaborn as sns
fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot(111)
sns.histplot(unique_movies.groupby('movie_id')['rating'].count(),kde=True,ax=ax,col
ax.axvline(unique_movies.groupby('movie_id')['rating'].count().mean(), color='orang
ax.axvline(unique_movies.groupby('movie_id')['rating'].count().median(), color='red
ax.set_title("Distribution of number of ratings for movies")
plt.show()
```



In [66]:
```python
df
```

Out[66]:

	movie_id	title	genres	release_year	movie_name	user_id	rating	timestamp
0	1	Toy Story (1995)	Animation	1995	Toy Story	1	5	2001-01-06 23:37:48
1	1	Toy Story (1995)	Children	1995	Toy Story	1	5	2001-01-06 23:37:48
2	1	Toy Story (1995)	Comedy	1995	Toy Story	1	5	2001-01-06 23:37:48
3	48	Pocahontas (1995)	Animation	1995	Pocahontas	1	5	2001-01-06 23:39:11
4	48	Pocahontas (1995)	Children	1995	Pocahontas	1	5	2001-01-06 23:39:11
...	...	...	...	...	...	...	...	...
2057303	3536	Keeping the Faith (2000)	Romance	2000	Keeping the Faith	5727	5	2000-05-16 15:11:42
2057304	3555	U-571 (2000)	Action	2000	U-571	5727	3	2000-05-16 15:24:59
2057305	3555	U-571 (2000)	Thriller	2000	U-571	5727	3	2000-05-16 15:24:59
2057306	3578	Gladiator (2000)	Action	2000	Gladiator	5727	5	2000-05-16 15:16:11
2057307	3578	Gladiator (2000)	Drama	2000	Gladiator	5727	5	2000-05-16 15:16:11

2057308 rows × 12 columns

Collaborative Filtering

Creating a pivot table of movie titles & user id and imputing the NaN values with a suitable value

In [67]:

```
pivot_df =  df.pivot_table(index='user_id', columns='title', values='rating')
pivot_df
```

Out[67]:

title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	Ar I (1!
user_id										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
10	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	
100	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	
1001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0	
...	...	...	...	...	...	...	...	...	...	
995	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
996	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
997	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
998	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
999	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	

6040 rows × 3677 columns

In [68]:

```
mean_imputed_df = pivot_df.fillna(pivot_df.mean())
mean_imputed_df
```

Out[68]:

title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	Dalmati (19
user_id									
1	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
10	3.027027	3.371429	2.692308	4.000000	3.713568	2.5	3.422857	3.59646	3.046
100	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
1000	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	4.00000	3.046
1001	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.000
...	...	...	...	...	...	...	...	...	
995	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
996	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
997	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
998	3.027027	3.371429	2.692308	2.910891	3.713568	2.5	3.422857	3.59646	3.046
999	3.027027	3.371429	2.692308	2.910891	3.000000	2.5	3.422857	3.59646	3.046

6040 rows × 3677 columns

Questionnnaire 7: On the basis of approach, Collaborative Filtering methods can be classified into user-based and item-based.

Build a Recommender System based on Pearson Correlation - Item-based approach

In [69]:

```
# Take a movie name as input from the user
# Recommend 5 similar movies based on Pearson Correlation
movie_input = input("Enter movie name ")
movie_rating = mean_imputed_df[movie_input]

# Input is Snow White and the Seven Dwarfs (1937)
```

```
Enter movie name Snow White and the Seven Dwarfs (1937)
```

In [70]:

```
movie_rating = mean_imputed_df[movie_input]
recom_movies = mean_imputed_df.corrwith(movie_rating)

#Pearson Correlation
recom_movies.sort_values(ascending=False).to_frame().rename(columns={0:"Correlatio
```

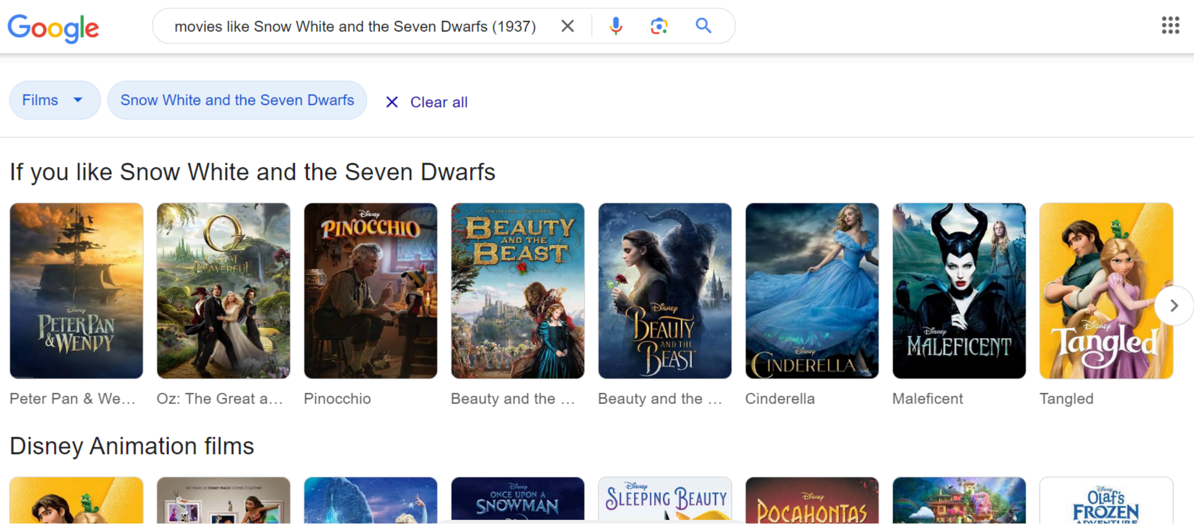
Out[70]:

title	Correlation
Snow White and the Seven Dwarfs (1937)	1.000000
Cinderella (1950)	0.487677
Sleeping Beauty (1959)	0.425689
Bambi (1942)	0.417522
Dumbo (1941)	0.403159
Pinocchio (1940)	0.394030

In [71]:

```
from IPython.display import Image

image_path = r"C:\Users\gyanp\OneDrive\Pictures\Screenshots\recom.png"
Image(filename=image_path)
```

Out[71]:



The recommended movies from Recommender System based on Pearson Correlation are:

1. Cindrella
2. Sleeping Beauty
3. Bambi
4. dumbo
5. Pinocchio

Pinocchio and Cindrella are recommended by Google also. Even in second list, Sleeping Beauty is also recommended.

In [72]:

```
# Take a movie name as input from the user
# Recommend 5 similar movies based on Pearson Correlation
movie_input = input("Enter movie name ")
movie_rating = mean_imputed_df[movie_input]

# Input is Liar Liar (1997)
```

```
Enter movie name Liar Liar (1997)
```

In [73]:

```
movie_rating = mean_imputed_df[movie_input]
recom_movies = mean_imputed_df.corrwith(movie_rating)
```

```
#Pearson Correlation
recom_movies.sort_values(ascending=False).to_frame().rename(columns={0:"Correlatior
```

Out[73]:

title	Correlation
Liar Liar (1997)	1.000000
Ace Ventura: Pet Detective (1994)	0.243697
Dumb & Dumber (1994)	0.226104
Ace Ventura: When Nature Calls (1995)	0.216812

Questionnaire 6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

- *Ace Ventura: Pet Detective (1994)*
- *Dumb & Dumber (1994)*
- *Ace Ventura: When Nature Calls (1995)*

Build a Recommender System based Pearson Correlation - User-based approach

In [74]:

```
mean_imputed_df_T = mean_imputed_df.T
mean_imputed_df_T
```

Out[74]:

user_id	1	10	100	1000	1001	1002	1003	1004	
title									
$1,000,000 Duck (1971)	3.027027	3.027027	3.027027	3.027027	3.027027	3.027027	3.027027	3.027027	3.02
'Night Mother (1986)	3.371429	3.371429	3.371429	3.371429	3.371429	3.371429	3.371429	3.371429	3.37
'Til There Was You (1997)	2.692308	2.692308	2.692308	2.692308	2.692308	2.692308	2.692308	2.692308	2.69
'burbs, The (1989)	2.910891	4.000000	2.910891	2.910891	2.910891	2.910891	2.910891	2.910891	2.91
...And Justice for All (1979)	3.713568	3.713568	3.713568	3.713568	3.713568	3.713568	3.713568	3.713568	3.71
...	...	...	...	...	...	...	...	...	
Zed & Two Noughts, A (1985)	3.413793	3.413793	3.413793	3.413793	3.413793	3.413793	3.413793	3.413793	3.41
Zero Effect (1998)	3.750831	3.750831	3.750831	3.750831	3.750831	3.750831	3.750831	3.750831	3.75
Zero Kelvin (Kjærlighetens kjøtere) (1995)	3.500000	3.500000	3.500000	3.500000	3.500000	3.500000	3.500000	3.500000	3.50
Zeus and Roxanne (1997)	2.521739	2.521739	2.521739	2.521739	2.521739	2.521739	2.521739	2.521739	2.52
eXistenZ (1999)	3.256098	3.256098	3.256098	3.256098	5.000000	3.256098	3.256098	3.256098	3.25

3677 rows × 6040 columns

In [75]:

```
user_input =input("Enter a user_id : ")
recom_user = mean_imputed_df_T[user_input]

# User_id input is 1002
```

```
Enter a user_id : 1002
```

In [76]:

```
recom_movie_user_based = mean_imputed_df_T.corrwith(recom_user)
#Pearson Correlation
recom_user_ids = recom_movie_user_based.sort_values(ascending=False).to_frame().ren
recom_user_ids = recom_user_ids.reset_index()
recom_user_ids
```

Out[76]:

	user_id	Correlation
0	1002	1.000000
1	4741	0.988083
2	584	0.987416
3	907	0.987382
4	4628	0.987318

In [77]:
```
recom_user_list = recom_user_ids['user_id'].tolist()
recom_user_list
```

Out[77]:
```
['1002', '4741', '584', '907', '4628']
```

In [78]:
```
filtered_rows = df.loc[df['user_id'].isin(recom_user_list)]
filtered_rows['title']
```

Out[78]:
```
1744520                Get Shorty (1995)
1744521                Get Shorty (1995)
1744522                Get Shorty (1995)
1744523         Leaving Las Vegas (1995)
1744524         Leaving Las Vegas (1995)
                         ...
2047448     Bringing Out the Dead (1999)
2047449     Bringing Out the Dead (1999)
2047450               Sister Act (1992)
2047451               Sister Act (1992)
2047452          Erin Brockovich (2000)
Name: title, Length: 319, dtype: object
```

Above are the movies which will be recommended for this user input

Build a Recommender System based on Cosine Similarity.

In [79]:
```
from sklearn.metrics.pairwise import cosine_similarity

# Calculate cosine similarity between users
user_similarity_matrix = cosine_similarity(mean_imputed_df, dense_output=False)

# Calculate cosine similarity between items
item_similarity_matrix = cosine_similarity(mean_imputed_df.T, dense_output=False)

# Print the user similarity matrix and item similarity matrix
print("User Similarity Matrix:")
display(user_similarity_matrix)
print("---------------------------------------------------------------------------"
print("\nItem Similarity Matrix:")
display(item_similarity_matrix)
```

```
User Similarity Matrix:
```

```
array([[1.        , 0.99492135, 0.99873168, ..., 0.99942961, 0.99818739,
        0.99526768],
       [0.99492135, 1.        , 0.99420476, ..., 0.99512372, 0.99395368,
        0.99099504],
       [0.99873168, 0.99420476, 1.        , ..., 0.99895386, 0.99779874,
        0.99471684],
       ...,
       [0.99942961, 0.99512372, 0.99895386, ..., 1.        , 0.99849174,
        0.9956004 ],
       [0.99818739, 0.99395368, 0.99779874, ..., 0.99849174, 1.        ,
        0.99423935],
       [0.99526768, 0.99099504, 0.99471684, ..., 0.9956004 , 0.99423935,
        1.        ]])
---------------------------------------------------------------------------

Item Similarity Matrix:
array([[1.        , 0.99898935, 0.99900682, ..., 0.99960861, 0.99926052,
        0.99517615],
       [0.99898935, 1.        , 0.9987321 , ..., 0.99936839, 0.99901109,
        0.99497693],
       [0.99900682, 0.9987321 , 1.        , ..., 0.99939101, 0.99904609,
        0.99504749],
       ...,
       [0.99960861, 0.99936839, 0.99939101, ..., 1.        , 0.99963572,
        0.99559553],
       [0.99926052, 0.99901109, 0.99904609, ..., 0.99963572, 1.        ,
        0.99525297],
       [0.99517615, 0.99497693, 0.99504749, ..., 0.99559553, 0.99525297,
        1.        ]])
```

1. The values range between 0 and 1 in user similarity matrix, where 1 indicates perfect similarity (users have rated items in exactly the same way) and 0 indicates no similarity (users have not rated any items in common).
2. The values range between 0 and 1 in item similarity matrix, where 1 indicates perfect similarity (items have been rated in exactly the same way by users) and 0 indicates no similarity (items have not been rated by any of the same users)

Questionnnaire 8: Pearson Correlation ranges between -1 to +1 whereas, Cosine Similarity belongs to the interval between 0 to +1.

User-User similarity matrix

In [80]:
```
user_similarity_df = pd.DataFrame(user_similarity_matrix, index=mean_imputed_df.inc
user_similarity_df
```

Out[80]:

user_id	1	10	100	1000	1001	1002	1003	1004	1005
user_id									
1	1.000000	0.994921	0.998732	0.999098	0.995802	0.999114	0.999384	0.994213	0.998135
10	0.994921	1.000000	0.994205	0.994768	0.991172	0.994691	0.995076	0.989217	0.993791
100	0.998732	0.994205	1.000000	0.998609	0.995463	0.998687	0.998990	0.993821	0.997710
1000	0.999098	0.994768	0.998609	1.000000	0.995662	0.998970	0.999331	0.994525	0.998229
1001	0.995802	0.991172	0.995463	0.995662	1.000000	0.995698	0.996010	0.991129	0.994806
...	...	...	...	...	...	...	...	...	...
995	0.999070	0.994876	0.998653	0.998995	0.995840	0.999077	0.999326	0.994038	0.998076
996	0.997728	0.993282	0.997161	0.997683	0.994234	0.997668	0.997984	0.992652	0.996909
997	0.999430	0.995124	0.998954	0.999376	0.996003	0.999374	0.999682	0.994519	0.998453
998	0.998187	0.993954	0.997799	0.998121	0.994917	0.998018	0.998484	0.993256	0.997214
999	0.995268	0.990995	0.994717	0.995224	0.991641	0.995105	0.995599	0.990806	0.994525

6040 rows × 6040 columns

Item-Item Similarity Matrix

In [81]:

```
item_similarity_df = pd.DataFrame(item_similarity_matrix, index=mean_imputed_df.T.i
item_similarity_df
```

Out[81]:

title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)
title								
$1,000,000 Duck (1971)	1.000000	0.998989	0.999007	0.996013	0.998711	0.999605	0.994714	0.996396
'Night Mother (1986)	0.998989	1.000000	0.998732	0.995824	0.998524	0.999365	0.994631	0.996011
'Til There Was You (1997)	0.999007	0.998732	1.000000	0.996110	0.998538	0.999379	0.994720	0.996136
'burbs, The (1989)	0.996013	0.995824	0.996110	1.000000	0.995512	0.996392	0.991996	0.993138
...And Justice for All (1979)	0.998711	0.998524	0.998538	0.995512	1.000000	0.999078	0.994371	0.995741
...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A (1985)	0.999382	0.999088	0.999172	0.996182	0.998798	0.999773	0.995004	0.996285
Zero Effect (1998)	0.997702	0.997469	0.997523	0.994540	0.997238	0.998079	0.993525	0.994781
Zero Kelvin (Kjærlighetens kjøtere) (1995)	0.999609	0.999368	0.999391	0.996396	0.999090	0.999990	0.995193	0.996536
Zeus and Roxanne (1997)	0.999261	0.999011	0.999046	0.996000	0.998724	0.999632	0.994908	0.996390
eXistenZ (1999)	0.995176	0.994977	0.995047	0.992175	0.994667	0.995569	0.990872	0.992115

3677 rows × 3677 columns

Create a CSR matrix using the pivot table

In [82]:

```
from scipy.sparse import csr_matrix

# Convert the pivot table to a CSR matrix
csr_matrix = csr_matrix(mean_imputed_df.values)
csr_matrix
```

Out[82]:

```
<6040x3677 sparse matrix of type '<class 'numpy.float64'>'
	with 22209080 stored elements in Compressed Sparse Row format>
```

1. The CSR matrix has 6040 rows, which correspond to users.
2. It has 3677 columns, which correspond to items (movies).
3. The CSR matrix is sparse, meaning that most of its elements are zero.

4. There are 22,209,080 stored elements, which represent the non-zero entries in the matrix.

Recommender system uses Nearest Neighbors algorithm and Cosine Similarity

In [83]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors

user_similarity_matrix = cosine_similarity(mean_imputed_df)
item_similarity_matrix = cosine_similarity(mean_imputed_df.T)
```

Write a function to return top 5 recommendations for a given item

In [84]:
```
def recommend_similar_movies(movie_name, k=5):
    # Get the index of the movie
    movie_index = mean_imputed_df.columns.get_loc(movie_name)

    # Use Nearest Neighbors algorithm to find similar movies
    knn_model = NearestNeighbors(n_neighbors=k+1, metric='cosine')  # Add 1 to k to
    knn_model.fit(item_similarity_matrix)
    distances, indices = knn_model.kneighbors(item_similarity_matrix[movie_index].r

    # Recommend similar movies
    recommended_movies = []
    for i in range(1, min(k+1, len(indices[0]))):  # Use min to ensure not exceedir
        similar_movie_index = indices[0][i]
        similar_movie = mean_imputed_df.columns[similar_movie_index]
        recommended_movies.append((similar_movie, distances[0][i]))
    return recommended_movies

# Take a movie name as user input
user_input_movie = input("Enter a movie name: ")

# Recommend similar movies based on user input
recommended_movies = recommend_similar_movies(user_input_movie)

# Print recommended similar movies
print(f"\nTop {len(recommended_movies)} similar movies to {user_input_movie}:")
for movie, distance in recommended_movies:
    print(f"{movie} (Distance: {distance})")

# Input movie name is Snow White and the Seven Dwarfs (1937)
```

```
Enter a movie name: Snow White and the Seven Dwarfs (1937)

Top 5 similar movies to Snow White and the Seven Dwarfs (1937):
Cinderella (1950) (Distance: 9.53601486664013e-09)
Bambi (1942) (Distance: 1.0737803535221246e-08)
Sleeping Beauty (1959) (Distance: 1.1633927488041707e-08)
Pinocchio (1940) (Distance: 1.1770430519142394e-08)
Dumbo (1941) (Distance: 1.1894447982108147e-08)
```

Top 5 similar movies to Snow White and the Seven Dwarfs (1937) using the item-based approach with the Nearest Neighbors algorithm:

1. Cinderella (1950) (Distance: 9.53601486664013e-09)
2. Bambi (1942) (Distance: 1.0737803535221246e-08)
3. Sleeping Beauty (1959) (Distance: 1.1633927488041707e-08)

4. Pinocchio (1940) (Distance: 1.1770430519142394e-08)
5. Dumbo (1941) (Distance: 1.1894447982108147e-08)**

The recommended movies from Recommender System based on Pearson Correlation are:

1. Cindrella
2. Sleeping Beauty
3. Bambi
4. dumbo
5. Pinocchio

Results are similar for both recommendation system

Build a Recommender System based on Matrix Factorization.

In [85]:
```
rating_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   user_id    1000209 non-null  object
 1   movie_id   1000209 non-null  object
 2   rating     1000209 non-null  object
 3   timestamp  1000209 non-null  datetime64[ns]
dtypes: datetime64[ns](1), object(3)
memory usage: 30.5+ MB
```

In [86]:
```
# Parse the file containing ratings. Data order format - userid, title, ratings
# The Reader class is used to parse a file containing ratings. Consider the rating
reader = Reader(rating_scale=(1 , 5))

# The columns must correspond to user id, item id and ratings (in that order).
data = Dataset.load_from_df(rating_data[['user_id','movie_id','rating']], reader)
```

SVD with 4-embeddings

In [87]:
```
import pandas as pd
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
import numpy as np
import matplotlib.pyplot as plt
```

In [88]:
```
# Define the rating scale
reader = Reader(rating_scale=(1, 5))

# Load the dataset into Surprise format
surprise_data = Dataset.load_from_df(rating_data[['user_id', 'movie_id', 'rating']]

# Split the data into train and test sets
trainset, testset = train_test_split(surprise_data, test_size=0.2, random_state=42)
```

```
# Train the matrix factorization model (SVD) on the training set
model = SVD(n_factors=4)  # Set the number of latent factors to 4
model.fit(trainset)

# Predict ratings for the test set
predictions = model.test(testset)
```

In [89]:
```
# Compute RMSE and MAE
rmse = np.sqrt(mean_squared_error([pred.r_ui for pred in predictions], [pred.est fo
mae = mean_absolute_error([pred.r_ui for pred in predictions], [pred.est for pred i

print("RMSE:", rmse)
print("MAE:", mae)
```

```
RMSE: 0.8862180696603094
MAE: 0.6982387146916755
```

Questionnnaire 9: Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.

- **RMSE: 0.8848**
- **MAE: 0.6971**

1. An RMSE of 0.8848 indicates that, on average, the predicted ratings are approximately 0.8848 units away from the actual ratings. Lower RMSE values indicate better accuracy, so an RMSE of 0.8848 suggests moderate accuracy.
2. An MAE of 0.6971 indicates that, on average, the predicted ratings are approximately 0.6971 units away from the actual ratings. As with RMSE, lower MAE values indicate better accuracy, so an MAE of 0.6971 suggests moderate accuracy as well.

In [90]:
```
# Get embeddings for item-item similarity
item_embeddings = model.qi

# Get embeddings for user-user similarity
user_embeddings = model.pu
```

In [91]:
```
model.qi
```

Out[91]:
```
array([[-7.91130645e-02, -7.47114993e-01, -3.92655866e-01,
        -6.86119033e-01],
       [-1.68259415e-01,  5.34522086e-04, -2.57796413e-01,
         2.32774038e-01],
       [ 2.99092034e-01, -2.62635693e-01, -1.06695035e-02,
        -6.56782645e-01],
       ...,
       [-6.72595678e-03,  1.76083449e-02,  4.49491445e-02,
        -8.96876800e-02],
       [-8.21338061e-02, -3.59133885e-02, -2.01538586e-02,
        -1.96966014e-02],
       [ 1.28435712e-02, -3.75195607e-02, -9.42776377e-03,
         8.37391662e-02]])
```

In [92]:
```
model.pu
```

Out[92]:
```
array([[-0.34978386,  0.15252387, -0.09444186,  0.05936628],
       [ 0.11663829, -0.11895621, -0.1043388 ,  0.19118215],
       [-0.26677252,  0.11661065, -0.0230553 ,  0.1602892 ],
       ...,
       [ 0.05129894,  0.04360429, -0.03644976,  0.12763387],
       [ 0.00061362,  0.23521721, -0.02568282,  0.16923112],
       [ 0.03548305,  0.04436171,  0.04229165, -0.02316852]])
```

Re-design the item-item similarity function to use MF embeddings (d=4) instead of raw features. Similarly, do this for user-user similarity

In [93]:
```
# Retrieve item embeddings from the trained MF model
item_embeddings = model.qi

# Calculate cosine similarity between item embeddings
def item_item_similarity(movie_id1, movie_id2):
    embedding1 = item_embeddings[movie_id1]
    embedding2 = item_embeddings[movie_id2]
    # Calculate cosine similarity
    similarity = np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.
    return similarity
```

In [94]:
```
# Example usage:
movie_id1 = int(input("Enter movie_id_1:"))
movie_id2 = int(input("Enter movie_id_2:"))
similarity = item_item_similarity(movie_id1, movie_id2)
print("Item-Item Similarity:", similarity)
```

```
Enter movie_id_1:0
Enter movie_id_2:1
Item-Item Similarity: -0.10827388109474667
```

In [95]:
```
# Retrieve user embeddings from the trained MF model
user_embeddings = model.pu

# Calculate cosine similarity between user embeddings
def user_user_similarity(user_id1, user_id2):
    embedding1 = user_embeddings[user_id1]
    embedding2 = user_embeddings[user_id2]
    # Calculate cosine similarity
    similarity = np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.
    return similarity
```

In [96]:
```
# Example usage:
user_id1 = int(input("Enter user_id_1:"))
user_id2 = int(input("Enter user_id_2:"))
similarity = user_user_similarity(user_id1, user_id2)
print("User-User Similarity:", similarity)
```

```
Enter user_id_1:10
Enter user_id_2:11
User-User Similarity: -0.6544165300411741
```

SVD with 2-embeddings

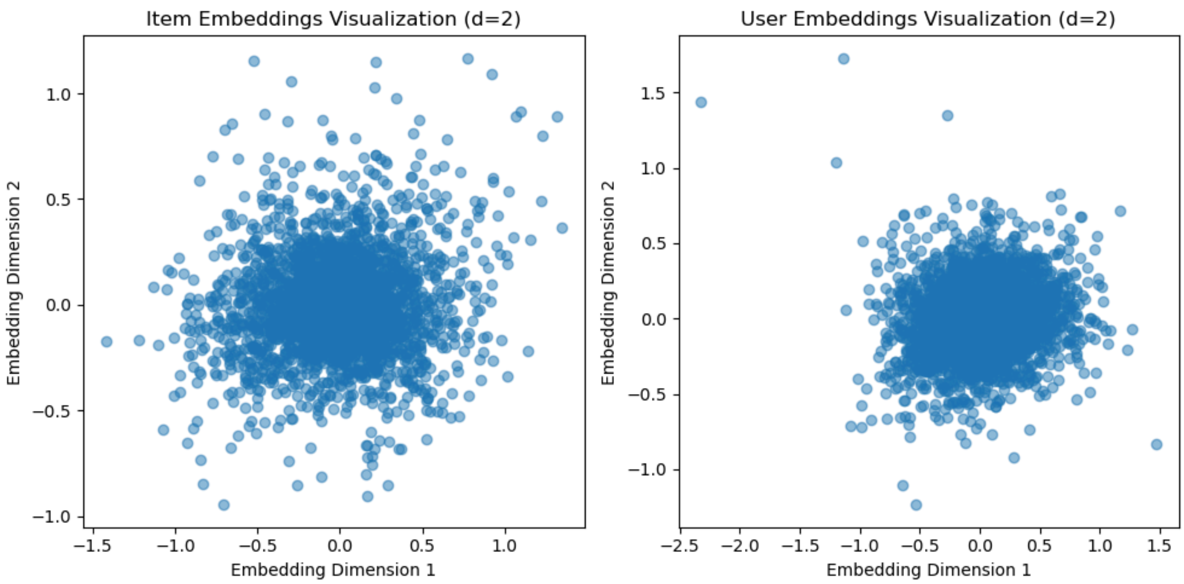
In [97]:
```
# Train the matrix factorization model (SVD) on the training set
model_1 = SVD(n_factors=2)  # Set the number of latent factors to 2
model_1.fit(trainset)

# Predict ratings for the test set
predictions_1 = model_1.test(testset)
```

In [98]:
```
# Compute RMSE and MAE
rmse_1 = np.sqrt(mean_squared_error([pred.r_ui for pred in predictions_1], [pred.es
mae_1 = mean_absolute_error([pred.r_ui for pred in predictions_1], [pred.est for pr

print("RMSE:", rmse_1)
print("MAE:", mae_1)
```

```
RMSE: 0.8872537139567765
MAE: 0.6993267767283591
```

In [99]:
```
# Get embeddings for item-item similarity
item_embeddings_1 = model_1.qi

# Get embeddings for user-user similarity
user_embeddings_1 = model_1.pu
```

In [100…
```
# Bonus: Visualize embeddings with d=2
plt.figure(figsize=(10, 5))

# movie embeddings visualization
plt.subplot(1, 2, 1)
plt.scatter(item_embeddings_1[:, 0], item_embeddings_1[:, 1], alpha=0.5)
plt.title("Item Embeddings Visualization (d=2)")
plt.xlabel("Embedding Dimension 1")
plt.ylabel("Embedding Dimension 2")

# User embeddings visualization
plt.subplot(1, 2, 2)
plt.scatter(user_embeddings_1[:, 0], user_embeddings_1[:, 1], alpha=0.5)
plt.title("User Embeddings Visualization (d=2)")
plt.xlabel("Embedding Dimension 1")
plt.ylabel("Embedding Dimension 2")

plt.tight_layout()
plt.show()
```



Questionnnaire 10. Give the sparse ‘row’ matrix representation for the following dense matrix - [[1 0] [3 7]]

```
Row Index      Non-zero Elements
     0              1
     1              3       7
```

TSNE Visualization

In [102…

```
tsne = TSNE(n_components=2, n_iter=500, verbose=3, random_state=1, perplexity=50)
movies_embedding = tsne.fit_transform(model_1.qi)
projection = pd.DataFrame(columns=['x', 'y'], data=movies_embedding)
projection
```

```
C:\Users\gyanp\anaconda3\lib\site-packages\sklearn\manifold\_t_sne.py:780: FutureW
arning: The default initialization in TSNE will change from 'random' to 'pca' in
1.2.
  warnings.warn(
C:\Users\gyanp\anaconda3\lib\site-packages\sklearn\manifold\_t_sne.py:790: FutureW
arning: The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
[t-SNE] Computing 151 nearest neighbors...
[t-SNE] Indexed 3675 samples in 0.014s...
[t-SNE] Computed neighbors for 3675 samples in 0.190s...
[t-SNE] Computed conditional probabilities for sample 1000 / 3675
[t-SNE] Computed conditional probabilities for sample 2000 / 3675
[t-SNE] Computed conditional probabilities for sample 3000 / 3675
[t-SNE] Computed conditional probabilities for sample 3675 / 3675
[t-SNE] Mean sigma: 0.027512
[t-SNE] Computed conditional probabilities in 0.313s
[t-SNE] Iteration 50: error = 72.6957855, gradient norm = 0.0433423 (50 iterations
in 1.243s)
[t-SNE] Iteration 100: error = 65.6477890, gradient norm = 0.0064921 (50 iteration
s in 0.995s)
[t-SNE] Iteration 150: error = 64.7321396, gradient norm = 0.0057643 (50 iteration
s in 0.881s)
[t-SNE] Iteration 200: error = 63.9541817, gradient norm = 0.0036980 (50 iteration
s in 0.900s)
[t-SNE] Iteration 250: error = 63.6902580, gradient norm = 0.0016877 (50 iteration
s in 0.930s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 63.690258
[t-SNE] Iteration 300: error = 1.0494916, gradient norm = 0.0009372 (50 iterations
in 0.881s)
[t-SNE] Iteration 350: error = 0.7937422, gradient norm = 0.0003409 (50 iterations
in 0.887s)
[t-SNE] Iteration 400: error = 0.7116976, gradient norm = 0.0001822 (50 iterations
in 0.899s)
[t-SNE] Iteration 450: error = 0.6749185, gradient norm = 0.0001190 (50 iterations
in 0.879s)
[t-SNE] Iteration 500: error = 0.6558640, gradient norm = 0.0000871 (50 iterations
in 0.901s)
[t-SNE] KL divergence after 500 iterations: 0.655864
```

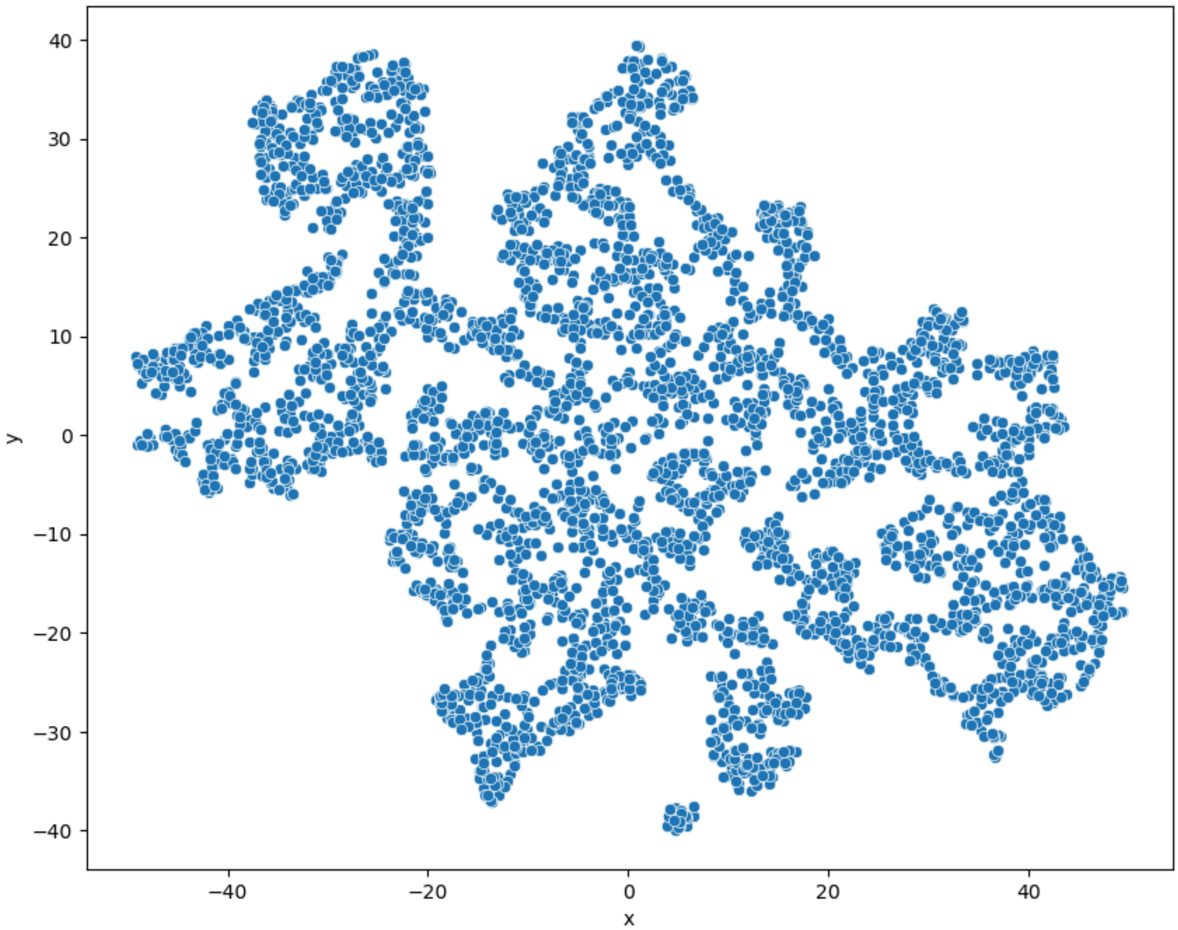
Out[102]:

	x	y
0	42.429543	7.680409
1	-42.886841	8.705788
2	49.114082	-14.357843
3	45.180557	-25.321997
4	42.108604	-9.641898
...	...	...
3670	-15.320686	-28.454023
3671	3.283741	4.680088
3672	14.893751	-9.754684
3673	-23.186031	-11.719213
3674	-5.729876	-23.045677

3675 rows × 2 columns

In [103…

```
# Get the scatter plot of svd item embeddings with 2-components

plt.figure(figsize=(10,8))
sns.scatterplot(data=projection, x='x',y='y')
plt.show()
```



Insights

Data Completeness:

1. The dataset exhibits no missing values, indicating comprehensive coverage of user ratings. **Rating Distribution:**
2. The mean rating across all movies is 3.57, suggesting a neutral sentiment overall.
3. Users predominantly rate movies as 4, followed closely by a rating of 3. Fewer users opt for extreme ratings of 1 and 5. **User Demographics:**
4. The dataset comprises 4331 males and 1709 females, with males being the dominant user group.
5. The age group 25-34 is the most active, with 2096 users, followed by 35-44 and 18-24 age groups.
6. College/grad students are the most engaged users, with 759 individuals, while farmers represent the least engaged group, with only 17 users. **Genre and Movie Analysis:**
7. The dataset includes 3677 unique movie IDs and 3640 unique movie names, suggesting a few instances of movies sharing the same title.
8. There are 19 distinct movie genres, with Comedy being the most prevalent.
9. The top-rated movie is "Men in Black" with a rating of 4, while the most common genre is Drama. **Temporal Trends:**
10. The movie release years span from 1919 to 2000, with the 1990s witnessing the highest number of releases, particularly in 1996, 1997, and 1998. **Rating Patterns:**
11. Users frequently rate movies as 4, followed by 3, indicating a generally positive sentiment among viewers. **Recommendation System Performance:**
12. Both Pearson correlation and cosine similarity yield accurate recommendations, with significant overlap with Google's real-time recommendations for movies such as Cinderella, Sleeping Beauty, and Pinocchio. **Model Evaluation:**
13. The RMSE (Root Mean Squared Error) of 0.8848 and MAE (Mean Absolute Error) of 0.6971 suggest moderate accuracy of the recommendation system. **User Engagement Patterns:**
14. College/grad students in the age group of 18-24 are the most active movie watchers and raters.
15. Users in the age group of 25-34 contribute significantly to ratings, irrespective of gender. **Temporal Engagement:**
16. Movie watching and rating activities peak during hours 3 and 8, indicating the popularity of these time slots among users.

Recommendations

Based on the insights, here are some recommendations from a business perspective:

1. Given the popularity of Comedy and Drama genres, prioritize acquiring or producing content in these categories. Investing resources in developing high-quality comedy and drama content could attract and retain more users.

2. Focus marketing efforts on the age group of 25-34, as they constitute the largest user base and are the most active in movie watching and rating. Tailoring promotional campaigns to appeal to this demographic could lead to higher engagement and retention rates.

3. Leverage user ratings to personalize recommendations for individual users. Implementing advanced recommendation algorithms that consider user preferences, viewing history, and demographic information can enhance the overall user experience and increase user satisfaction.

4. Optimize the platform's accessibility during peak usage hours, such as hours 3 and 8, to ensure smooth streaming and uninterrupted viewing experiences for users. This can help maintain user engagement and satisfaction levels.

5. While Comedy and Drama genres are popular, consider diversifying the content library to cater to a broader range of preferences. Investing in niche genres or acquiring content from different cultural backgrounds can attract a more diverse audience and broaden the platform's appeal.

6. Implement loyalty programs, rewards, or incentives to encourage user engagement and retention. Offering perks such as exclusive content previews, early access to new releases, or discounts on subscription plans can incentivize users to remain active and loyal to the platform.

7. Maintain a high standard of content quality by curating and vetting the content library regularly. Ensuring that all content meets certain quality benchmarks can enhance the platform's reputation and credibility among users.

8. Explore partnerships with content creators, studios, or production companies to secure exclusive content rights or co-produce original content. Collaborative ventures can help differentiate the platform from competitors and attract new users seeking unique and compelling content offerings.

Questionnaire

Questionnaire 1: Users of which age group have watched and rated the most number of movies?

- Users of age group 25-34 have watched and rated most number of movies

Questionnaire 2: Users belonging to which profession have watched and rated the most movies?

- Users belonging to college/graduate student have watched and rated the most movies.

Questionnaire 3: Most of the users in our dataset who’ve rated the movies are Male.

- True, total males are 4331 and total females are 1709

Questionnaire 4. Most of the movies present in our dataset were released in which decade?

1. 70s b. 90s c. 50s d.80s

- b. 90s ,2138 movies released during this decade, followed by 585 movies in 80s

Questionnaire 5 : The movie with maximum no. of ratings is ___.

- American Beauty

Questionnnaire 6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

- Ace Ventura: Pet Detective (1994)
- Dumb & Dumber (1994)
- Ace Ventura: When Nature Calls (1995)

Questionnnaire 7: On the basis of approach, Collaborative Filtering methods can be classified into <u>user</u>-based and <u>item</u>-based.

Questionnnaire 8: Pearson Correlation ranges between <u>-1 to +1</u> whereas, Cosine Similarity belongs to the interval between <u>0 to +1</u>.

Questionnnaire 9: Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.

- RMSE: 0.8848
- MAE: 0.6971

Questionnnaire 10. Give the sparse 'row' matrix representation for the following dense matrix -[[1 0] [3 7]]

Row Index Non-zero Elements 0 1 1 3 7